SISTEMAS DIGITALES

TEMA 2:

Introducción a los lenguajes de descripción hardware de alto nivel

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS (I)

En este capítulo vamos a hacer una introducción a los lenguajes de descripción hardware y más concretamente el entorno VERIBEST, con el objetivo de poder realizar modelos compilables y simulables en VHDL, de diferentes unidades funcionales de los computadores, elementos estos que se introducen a lo largo de la materia de Sistemas Digitales.

Las ventajas de emplear herramientas de descripción hardware es la portabilidad de los diseños entre diferentes herramientas y la independencia entre diseño y tecnología.

En este curso, dado que SSDD es una materia básica, vamos a utilizar solamente una parte reducida de las posibilidades que tiene la herramienta, centrándonos en la parte de simulación de circuitos digitales, sin utilizar las posibilidades de síntesis de los mismos.

1. INTRODUCCIÓN Y OBJETIVOS (II)

V: VHSIC: Very High Speed Integrated Circuits
H: Hardware
D: Description
L: Language

- IEEE estándar 1076-1987 (1076-1993) (1076-2001)
- Desarrollado a partir de lenguaje ADA → Lenguaje concurrente
- Recomendado por el Departamento de Defensa de USA

1. INTRODUCCIÓN Y OBJETIVOS (III)

HISTORIA

- **Agosto 1985**. primer borrador de VHDL presentado por Intermitrics, IBM y Texas Instruments.

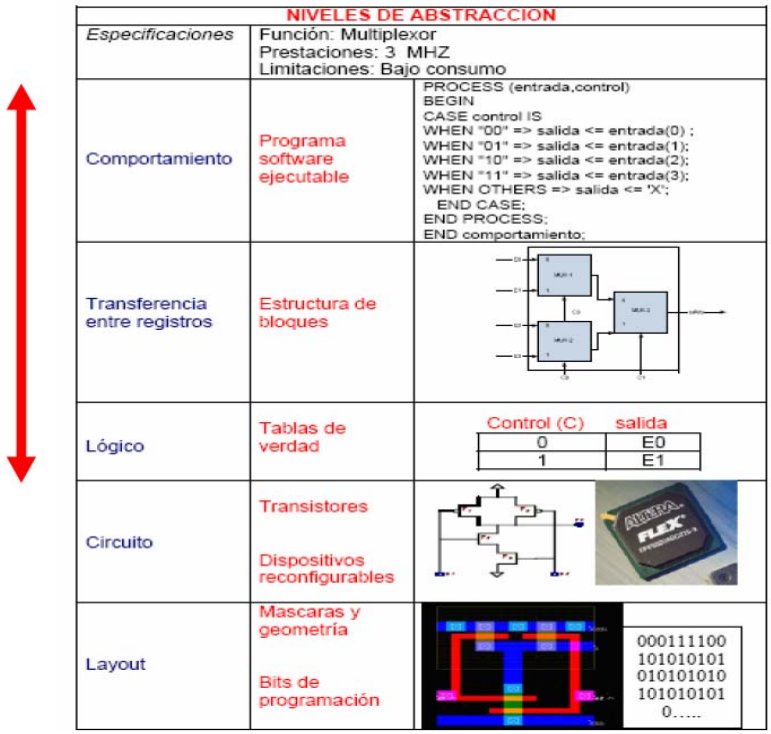
- **Diciembre 1987**. Se aprobó como estándar IEEE-1076-87 (VHDL-87)
  - Consolidación como herramienta de aplicación en la industria electrónica.

- **Año 1993**. Aprobación de una nueva versión del estándar con incorporación de nuevas características y definición de nuevas librerías estándar.
  - VHDL-93

- **Año 2001**. Última revisión (VHDL-2001)

1. INTRODUCCIÓN Y OBJETIVOS (IV)

NIVELES DE ABSTRACCIÓN (I)



1. INTRODUCCIÓN Y OBJETIVOS (V)

NIVELES DE ABSTRACCIÓN (II)

NIVELES DE ABSTRACCIÓN	DESCRIPCIÓN/EJEMPLO
Sistema	Descripción de un sistema y de sus prestaciones/ Computadores, impresoras, unidades de disco
Chip	Circuito que cumple con las especificaciones del sistema/ Microprocesadores, memorias, puertos de E/S….
Transferencia entre registros	Descripción de los flujos de datos/ Conjunto de registros y buses de interconexión
Lógica	Tabla de verdad y ecuaciones lógicas que describen un sistema/Implementación a través de bloques combinacionales y secuenciales
Circuito eléctrico	Componentes eléctricos/ Transistores, resistencias, capacidades,….
Físico	Descripción geométrica de elementos y compuestos químicos/ Máscaras con metal, silicio policristalino, área activa. NOTA: En este nivel no puede expresarse comportamiento

1. INTRODUCCIÓN Y OBJETIVOS (VI)

NIVELES DE ABSTRACCIÓN (III)

- NIVELES DE DISEÑO CUBIERTOS POR VHDL:
  - SISTEMA: solamente una pequeño parte
  - CHIP
  - TRANFERENCIA ENTRE REGISTROS
  - CIRCUITO LÓGICO
  - CIRCUITO ELÉCTRICO: no cubre la totalidad

- VHDL permite mezclar en una misma descripción diferentes niveles de abstracción.

- El tratamiento concurrente del lenguaje permite realizar descomposiciones jerárquicas.
  - Modelos estructurales en los que los componentes pueden ser descritos a partir de otros modelos estructurales más sencillos

1. INTRODUCCIÓN Y OBJETIVOS (VII)

CARACTERÍSTICAS

- Su principal dominio de aplicación es el modelado de dispositivos hardware, con el objetivo de la comprobación de su funcionalidad.
- Es un lenguaje con una semántica orientada a la simulación.
- Modelo de simulación por EVENTOS
- Metodología de diseño *top-down*

1. INTRODUCCIÓN Y OBJETIVOS (VIII)

- CONCURRENCIA: Actividades concurrentes son *sucesos que ocurren en paralelo*. El hardware es por definición concurrente (conjunto de bloques funcionado en paralelo) y un evento puede disparar varios procesos al mismo tiempo. El elemento básico que ofrece VHDL para modelar paralelismo es el proceso (PROCESS).

- ESTRUCTURA: *Ordenamiento de bloques en una jerarquía*. Cada bloque se puede describir en estilo RTL (REGISTER TRANSFER LEVEL), comportamiento o mixto.

- SECUENCIA: Las sentencias secuenciales se ejecutan *una después de otra*, como en lenguajes de software con un solo microprocesador.

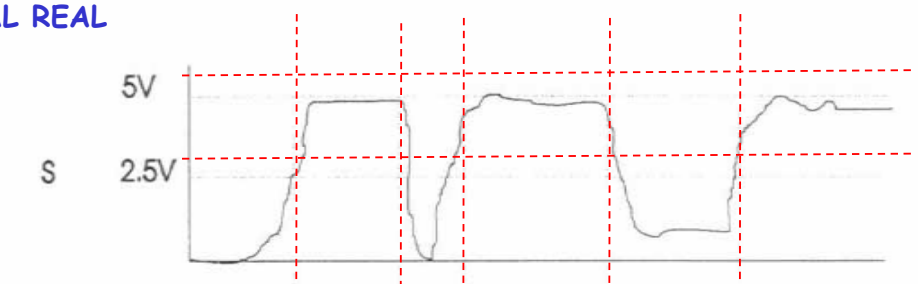
- TIEMPO: VHDL permite modelar el concepto de tiempo. *Simulación está dirigida por eventos*. Un evento es producido por un cambio en una señal en un determinado tiempo de simulación. La respuesta de un modelo a un evento puede provocar nuevos eventos.

2. CARACTERIZACIÓN DE LAS SEÑALES (I)

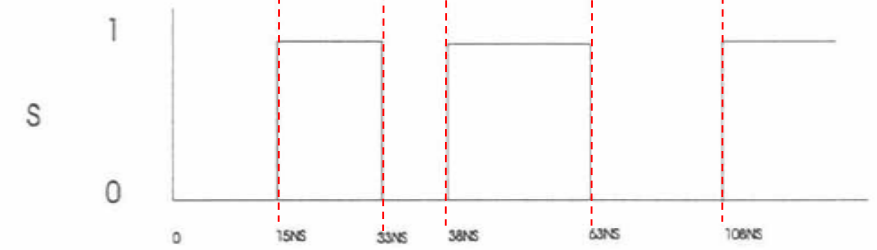
UNA SEÑAL SE PUEDE REPRESENTAR EN UNOS EJES TIEMPO/TENSIÓN

EJEMPLO

SEÑAL REAL

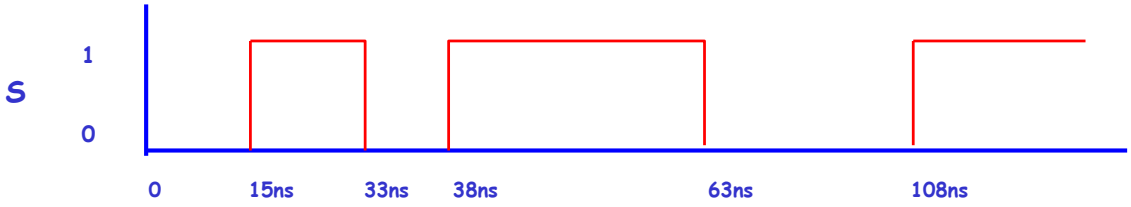


SEÑAL IDEALIZADA



2. CARACTERIZACIÓN DE LAS SEÑALES (II)

TRANSACCIONES



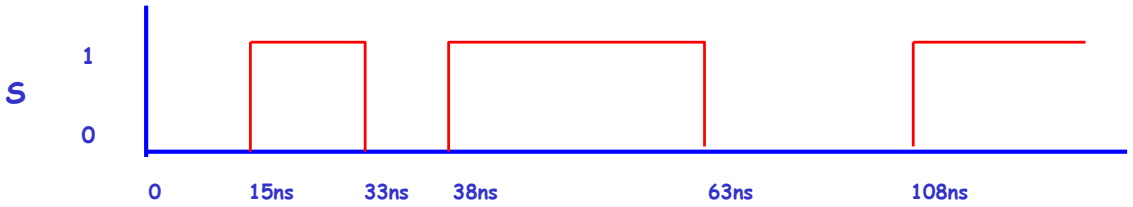
UNA SEÑAL TIENE UN VALOR EN UN ESPACIO TEMPORAL DETERMINADO

TRANSACCIÓN: PAR (VALOR, TIEMPO)

(0, 0 ns) (0, 7 ns) (1, 15 ns) (1, 30 ns) (0, 33 ns) (1, 38 ns)
(0, 63 ns) (1, 108 ns)

2. CARACTERIZACIÓN DE LAS SEÑALES (III)

EVENTOS

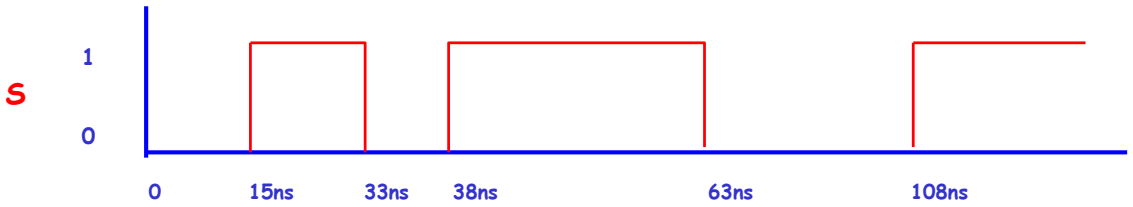


EVENTO: CÁMBIO DE VALOR DE UNA SEÑAL EN UN TIEMPO DETERMINADO

(0, 0 ns) (1, 15ns) (0, 33 ns) (1, 38 ns) (0, 63 ns) (1, 108 ns)

2. CARACTERIZACIÓN DE LAS SEÑALES (IV)

PROPAGACIÓN DE LAS SEÑALES

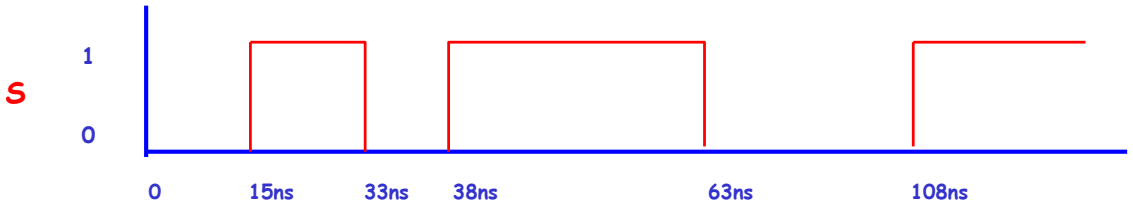


PROPAGACIÓN: EVOLUCIÓN TEMPORAL ASCENTE DE LAS TRANSACCIONES/EVENTOS DE UNA SEÑAL

```
S <= '0','1'AFTER 15 ns, '0'AFTER 33 ns, '1'AFTER 38 ns,
     '0'AFTER 63 ns, '1'AFTER 108 ns;
```

2. CARACTERIZACIÓN DE LAS SEÑALES (V)

DRIVERS (I)



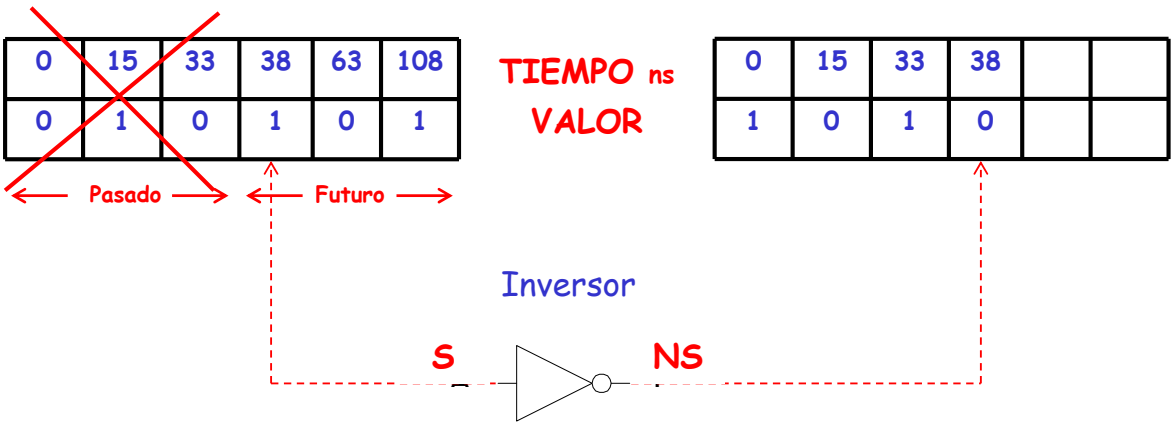
DRIVER: ES UNA COLA (TABLA) DE TRANSACCIONES QUE ALMACENA LA FORMA DE ONDA DE LA SEÑAL

TIEMPO	0	15	33	38	63	108
VALOR	0	1	0	1	0	1

En el tiempo de simulación 0ns → el driver tiene el valor 0
En el tiempo de simulación 38ns → el driver tiene el valor 1 . . . .

2. CARACTERIZACIÓN DE LAS SEÑALES (VI)

DRIVERS (II)



Ns<= NOT s;

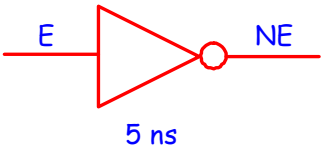
Al avanzar el tiempo de simulación las transacciones procesadas de eliminan de la cola

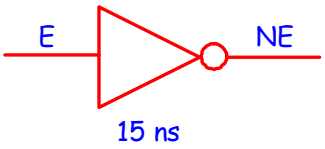
TIEMPO actual de simulación 38ns

3. RETARDOS EN LAS SEÑALES (I)

DEFINICIÓN Y TIPOS (I)

DEFINICIÓN: retardo es el tiempo que tarda en propagarse una señal desde un punto de un circuito a otro.





NE<= NOT E AFTER 5 ns; NE<= NOT E AFTER 15 ns;

•Ambos circuitos realizan la misma función, la operación, NOT; pero uno de ellos tiene un retardo de 5 ns, por tanto es más rápido que el otro, cuyo retardo es de 15 ns.

TIPOS: Delta (ð)
Inercial
De Transporte

3. RETARDOS EN SEÑALES (II)

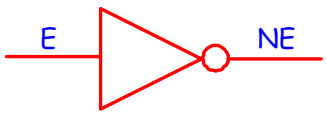
DELTA (I)

DELTA: Es el retardo infinitesimal, siempre despreciable respecto al tiempo de simulación, que separa la excitación de la respuesta en un sistema.

Es el tiempo que tarda en propagarse una señal desde la entrada hasta la salida de un circuito.

Cuando no se especifica un retardo para una señal, se toma por defecto el retardo delta

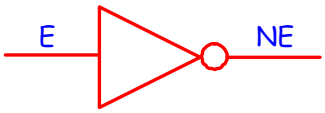
EJEMPLO:

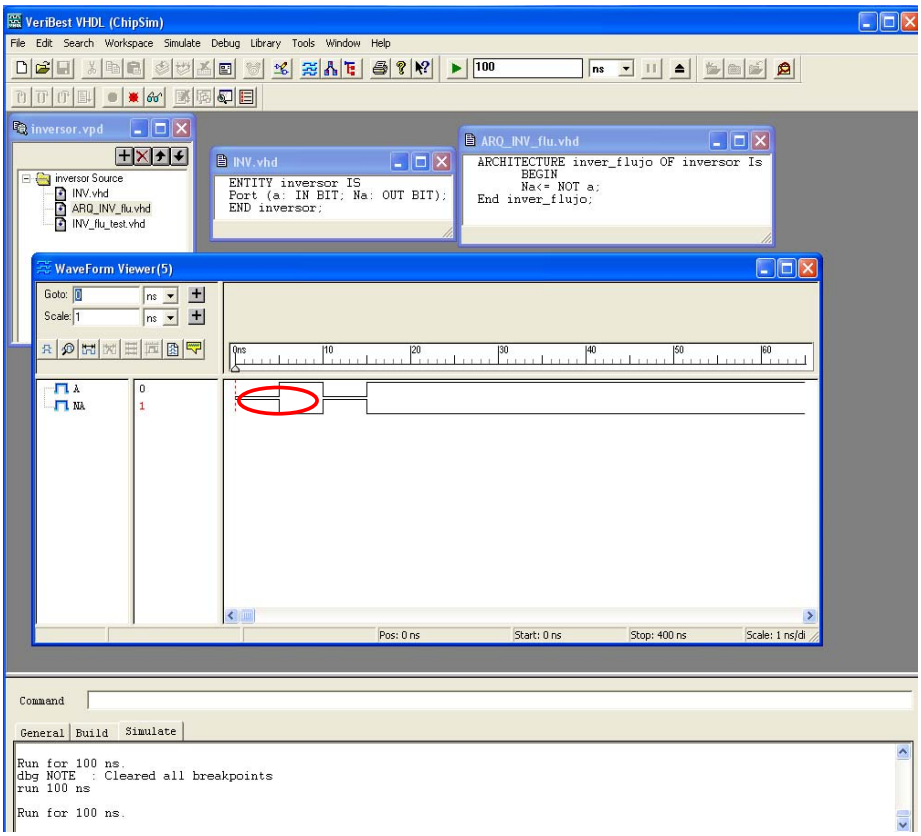


NE<= NOT E;

3. RETARDOS EN SEÑALES (III)

DELTA (II)





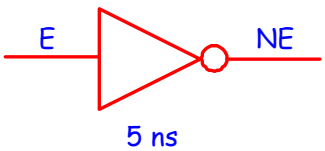
3. RETARDOS EN SEÑALES (IV)

INERCIAL (I)

El retardo Inercial modela el comportamiento temporal de la conmutación de los circuitos

Señal<= [expresión] señal **AFTER** tiempo;

EJEMPLOS:

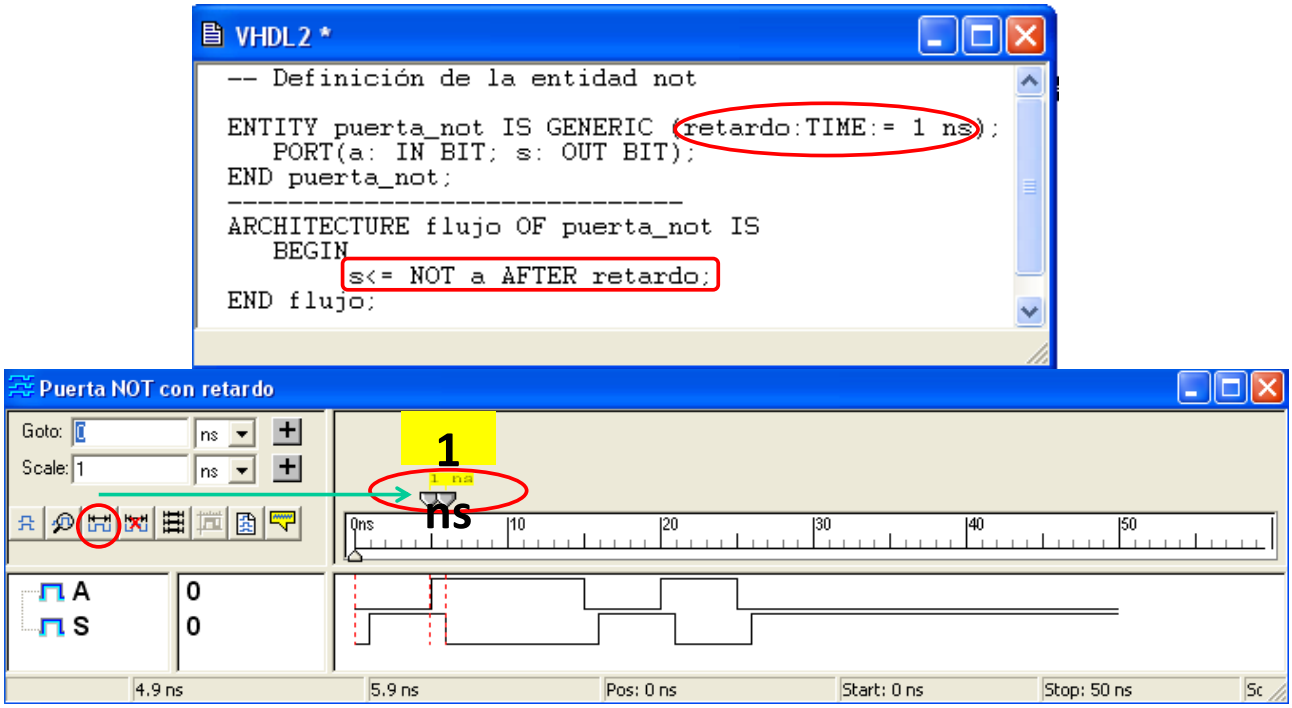


NE<= **NOT** E **AFTER** 5 ns;

S <= a **AND** b **AFTER** 15 ns;
S <= (**NOT** (a **AND** b) **OR** (c **OR** b)) **AFTER** 25 ns;
Cr<= c **AFTER** 35 ns;

3. RETARDOS EN SEÑALES (IV)

INERCIAL (II)



4. UNIDADES DE DISEÑO (I)

JERARQUÍAS EN VHDL (I)

- **Un diseño en VHDL consiste en una jerarquía de componentes compilados**
  - Los componente están implementados por arquitecturas, siguiendo uno de los tres estilos de descripción:
    - COMPORTAMIENTO
    - FLUJO DE DATOS
    - ESTRUCTURAL

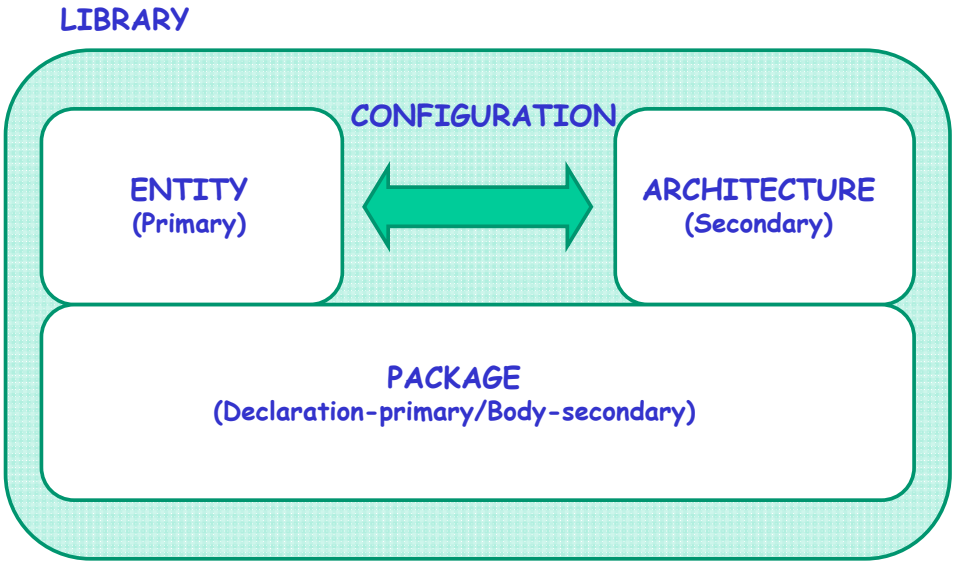
- **Una vez que los diseños están compilados, se ubican en una biblioteca y pueden utilizarse como componentes en los siguientes diseños.**

- **Los componentes tiene que ser declarados antes de realizar una instancia de ellos desde la biblioteca.**

4. UNIDADES DE DISEÑO (II)

JERARQUÍAS EN VHDL (II)

- Un diseño que contenga componentes requiere establecer una configuración que asocie entidades con su arquitectura.



4. UNIDADES DE DISEÑO (III)

JERARQUÍAS EN VHDL (III)

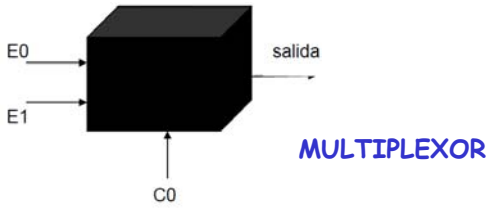
ELEMENTOS DE LA LIBRERÍA	DEFINICIONES
ENTIDAD	REPRESENTA LA ESPECIFICACIÓN DE LA INTERFAZ DE UN CIRCUITO; ES DECIR, SUS ENTRADAS Y SALIDAS Es análoga a un símbolo en un esquemático
ARQUITECTUTA	REPRESENTA LA FUNCIONALIDAD O LA ESTRUCTURA DE UNA ENTIDAD Puede ser descrita en tres niveles: nivel algorítmico, RTL o estructural
PAQUETE	AGRUPA A UN CONJUNTO DE DECLARACIONES Tipos, constantes, señales, variables, ficheros, variables compartidas, componentes, atributos,….
CUERPO DE PAQUETE	AGRUPA LA DEFINICIÓN DE LOS SUBPROGRAMAS Y LAS CONSTANTES DIFERIDAS DECLARADAS EN EL PAQUETE
CONFIGURACIÓN	ASOCIA COMPONENTES Y ARQUITECTURAS CON ENTIDADES

4. UNIDADES DE DISEÑO (IV)

ENTIDAD (ENTITY) (I)

Define el nombre de un componente y su interfaz de entrada-salida (señales de entrada salida, tipo de datos, entorno de uso)

- Modelo de caja negra



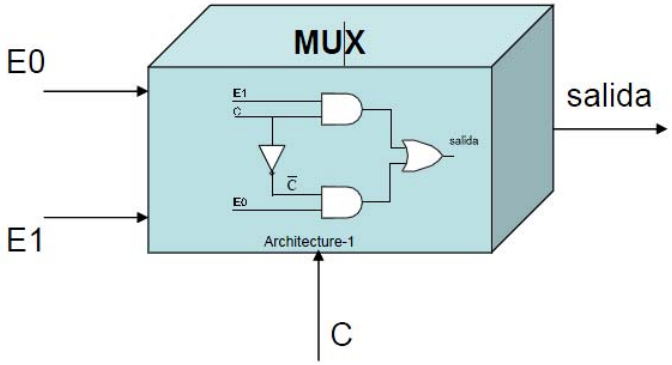
- Un componente puede tener la complejidad que se desee
  - puerta, sumador, ALU, computador….
- No pueden existir dos entidades con el mismo nombre

4. UNIDADES DE DISEÑO (V)

ENTIDAD (ENTITY) (II)

- La funcionalidad de la entidad se define a través de la arquitectura

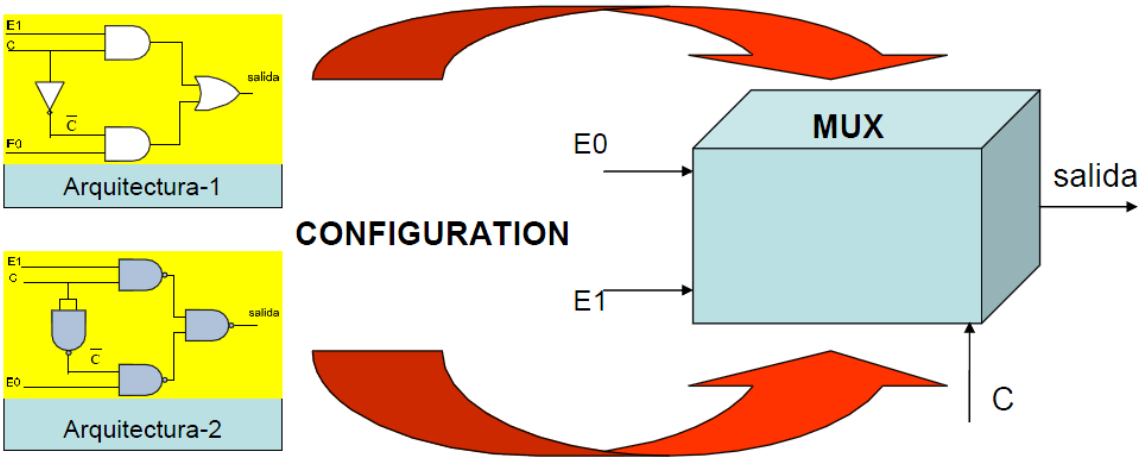
MULTIPLEXOR: construido con puertas lógicas básicas (inversores, AND y NOR)



4. UNIDADES DE DISEÑO (VI)

ENTIDAD (ENTITY) (III)

- Una entidad puede tener asociadas más de una arquitectura



4. UNIDADES DE DISEÑO (VII)

FORMATO DE LA ENTIDAD (IV)

```
ENTITY nombreEntidad IS
        [GENERIC (lista)];
        [PORT (lista)];
        [declaraciones de constantes, variables, señales,..]
        ……..
END nombreEntidad;
```

EJEMPLO:

```
ENTITY puerta_and2 IS
……….
……..
END puerta_and2;
```

Puerta AND

- palabra reservada
- [ ] indica sentencias opcionales

4. UNIDADES DE DISEÑO (VIII)

OPCIONES DE LAS ENTIDADES (V)

PORT (nombre: nombre_E/S tipoDato);

Todas las entidades TIENEN que incluir PORT con la EXCEPCIÓN de las entidades de TEST (se utilizan para testear las arquitecturas)

- **nombre se refiere a una señal**
  - Es conveniente que esté relacionado con la función que realizan, número de entradas, ……
- **nombre_E/S se refiere al modo de la señal:**
  - **IN: entrada**
  - **OUT: salida**
  - **INOUT: entrada y salida**
  - **BUFFER: salida con realimentación interna**

4. UNIDADES DE DISEÑO (IX)

OPCIONES DE ENTIDADES (VI)

- **tipoDato se refiere a uno de los tipos de datos predefinidos**

  o Paquete estándar (por defecto) no hay que declararlo

  - **BOOLEAN: *true, false***
  - **CHARACTER**
  - **REAL**
  - **BIT: ´0´,´1´**
  - **TIME**

  o Otros paquetes hay que declararlos previamente

```
LIBRARY ieee
USE ieee.std_logic_1164.all;
```

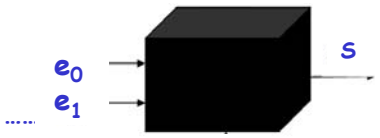
4. UNIDADES DE DISEÑO (X)

OPCIONES DE LAS ENTIDADES (VII)

EJEMPLO: entidad de AND con PORT

```
ENTITY and_2 IS
        PORT (e0,e1: IN BIT; s: OUT BIT);
END and_2;
```

```
ARCHITECTURE and_2_flujo OF and_2 IS
BEGIN
        S<= (e0 AND e1);
END and_2_flujo;
```



Puerta AND

- palabra reservada
- En este ejemplo la herramienta utiliza por defecto la librería estándar

4. UNIDADES DE DISEÑO (XI)

OPCIONES DE LAS ENTIDADES (VIII)

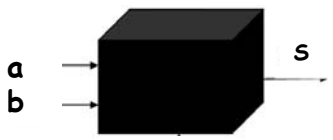
GENERIC (nombre: tipoDato:= [valorinicial]);

Pasa un dato específico a un componente

Se usa la librería `std_logic_1164`

EJEMPLO: entidad de AND con GENERIC

```
LIBRARY ieee;
USE ieee std_logic_1164 all;
-- se tiene que citar la librería
ENTITY and_2 IS
    GENERIC (retardo: TIME:=1ns);
    PORT (e0, e1: IN STD_LOGIC; s: OUT STD_LOGIC);
END and_2;
```



Puerta xor

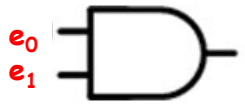
```
ARCHITECTURE and_2_flujo OF and_2 IS
BEGIN
        S<= (e0 AND e1) AFTER retardo;
END and_2_flujo;
```

4. UNIDADES DE DISEÑO (XII)

OPCIONES DE LAS ENTIDADES (IX)

EJEMPLO: tipo de dato

AND



tipo de datos: standard

and2_flujo

```
-- Definicion de la entidad and2

ENTITY and2 IS
   PORT(e0,e1:IN BIT; s:OUT BIT);
END and2;
```

NO es necesario declarar ninguna librería

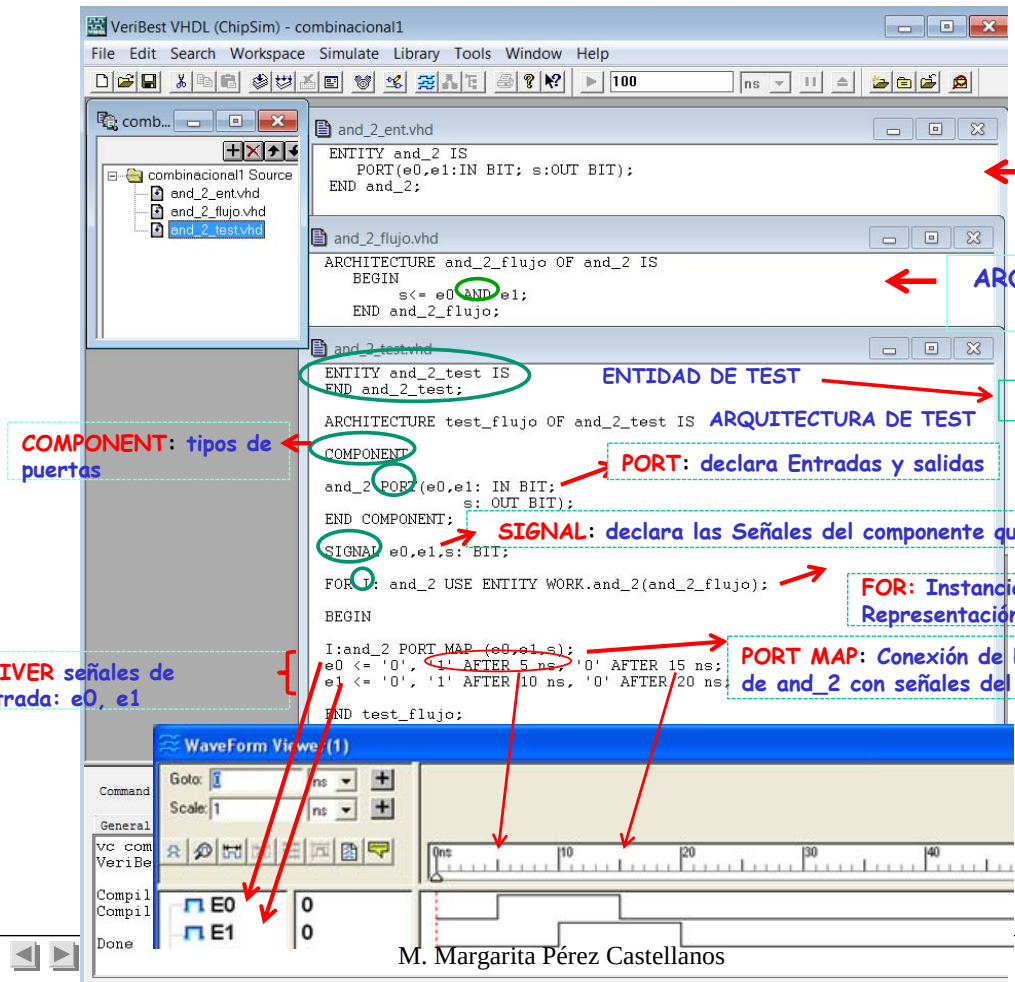
tipo de datos: standard_logic_1164

VHDL1 *

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Definicion de la entidad and2

ENTITY and2 IS
   PORT(e0,e1:IN STD_LOGIC;
        s:OUT STD_LOGIC);
END and2;
```

OBLIGATORIO: declarar la librería y el paquete de los que se hace uso

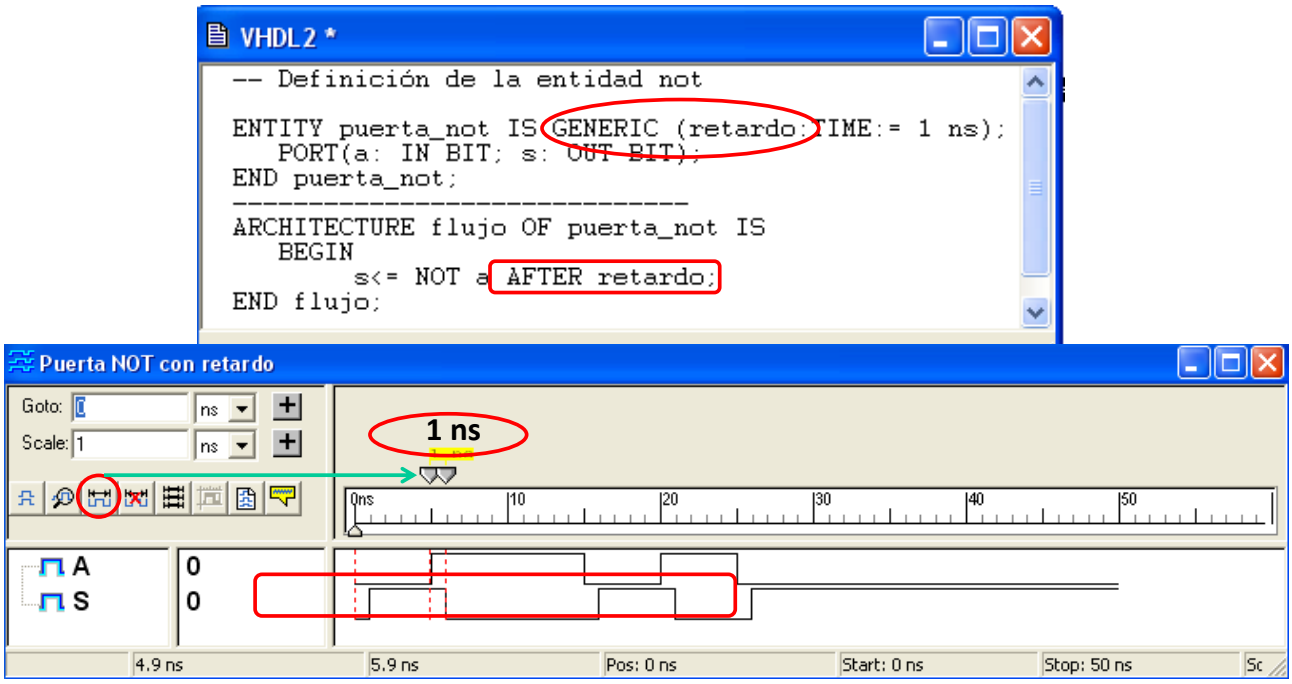
4.Unidades de Diseño (XIII)



4. UNIDADES DE DISEÑO (XIV)

OPCIONES DE LAS ENTIDADES (XI)

Visualización del retardo inercial (ver transparencias 17-21)



4. UNIDADES DE DISEÑO (XV)

ARQUITECTURA (I)

- Una arquitectura define la funcionalidad de la entidad a la que está asociada.

- Describen las operaciones que se efectúan sobre las entradas de la entidad y que determinan el valor de sus salidas en cada momento.

- Una entidad puede tener más de una modelo de arquitectura

4. UNIDADES DE DISEÑO (XVI)

FORMATO DE LA ARQUITECTURA (II)

ARCHITECTURE nombreArchitectura **OF** nombreentidad **IS**
[constantes, variables, señales intermedias de interconexión, tipos, componentes…..]

BEGIN

[PROCESS sentencias secuenciales] → SENTENCIAS SECUENCIALES
[asignación concurrente de señal]
[llamada concurrente a procedimientos]
[llamada concurrente a funciones]
[Instancia DE COMPONENTES]
ASSERT
GENERATE
BLOCK

SENTENCIAS SECUENCIALES
WAIT
ASSERT
Asignación secuencial de señal,
Asignación de variable
Llamada secuencial a funciones
Llamada secuencial a procedimientos
IF, CASE, LOOP, NEXT, EXIT, RETURN, NULL, ASSERTION

END nombreArquitectura;

4. UNIDADES DE DISEÑO (XVII)

ARQUITECTURA:ESTILOS DE DESCRIPCIÓN (I)

- Independientemente del nivel de abstracción, la descripción del modelo se puede realizar siguiendo las siguientes estilos:
  - COMPORTAMIENTO (Behavioral)
  - FLUJO DE DATOS o TRANSFERENCIA ENTRE REGISTROS (Data Flow)
  - ESTRUCTURAL (Structural)
- En VHDL un sistema puede mezclar diferentes estilos a la hora de describir sus diferentes componentes.

4. UNIDADES DE DISEÑO (XVIII)

ARQUITECTURA:ESTILOS DE DESCRIPCIÓN (II)

COMPORTAMIENTO, ALGORÍTMICO O SECUENCIAL (*BEHAVIORAL*)

- En este estilo se modela la funcionalidad del componente por medio de los recursos algorítmicos del lenguaje.

- Se describe el algoritmo que refleja el comportamiento de dicho componente.

- No tiene ningún significado hardware

4. UNIDADES DE DISEÑO (XIX)

ARQUITECTURA:ESTILOS DE DESCRIPCIÓN (III)

FLUJO DE DATOS O TRANSFERENCIA ENTRE REGISTROS (*DATA FLOW*)

- Se especifican los flujos de datos del sistema y la interconexión entre sus componentes

- El proceso de descripción se realiza por medio de funciones lógicas que se ejecutarán de forma concurrente.

- Tiene un cierto significado no detallado de la implementación hardware

4. UNIDADES DE DISEÑO (XX)

ARQUITECTURA:ESTILOS DE DESCRIPCIÓN (IV)

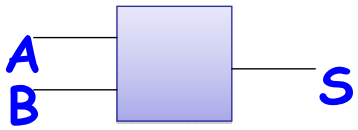
ESTRUCTURAL O LÓGICO (*STRUCTURAL*)

- Se definen o instancian todas las partes del sistema y sus interconexiones.
- Es la descripción de los bloque con dispositivos lógicos.
- Implementación hardware detallada
- Resulta muy útil cuando para aprovechardiseños compilados y guardados en bibliotecas de componentes.

4. UNIDADES DE DISEÑO (XXI)

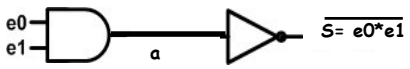
Representación/especificación o modelado de un circuito combinacional con 2 entradas y una salida S es cualquier función lógica de A y B (AND, OR, NAND, NOR, XOR, …)

Como bloque funcional → Solo se especifican las señales de entrada y salida, pero no su funcionalidad



Como función lógica → S=f(A,B). Se describe el flujo de datos

Como conjunto de componentes conectados con señales. Se describe la estructura del circuito con elementos hardware



Como tabla de verdad → valores binarios de la salida para cada una de las combinaciones de las entradas. Se describe el comportamiento del circuito

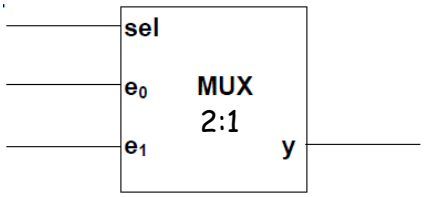
A	B	S
0	0	*
0	1	*
1	0	*
1	1	*

4. UNIDADES DE DISEÑO (XXII)

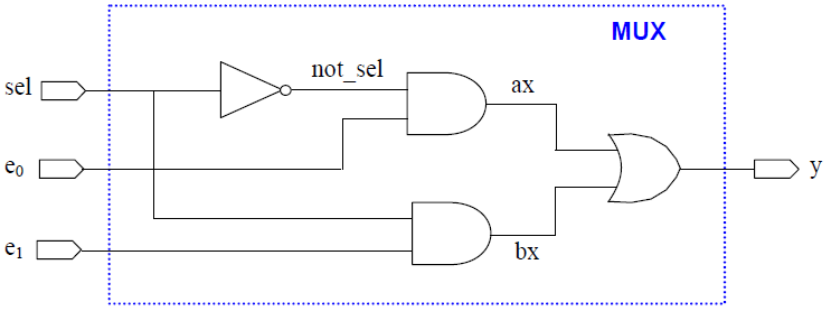
EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (I)

Descripción VHDL de un multiplexor 2:1

Entradas de datos: e_0 y e_1,
Señal de control: **sel**
Salida: **y**



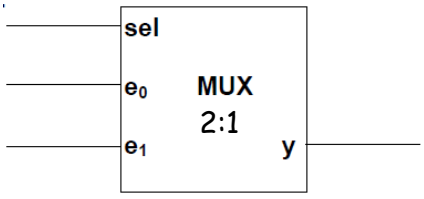
Se puede construir con puertas lógicas según la figura:



4. UNIDADES DE DISEÑO (XXIII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (II)

ENTIDAD



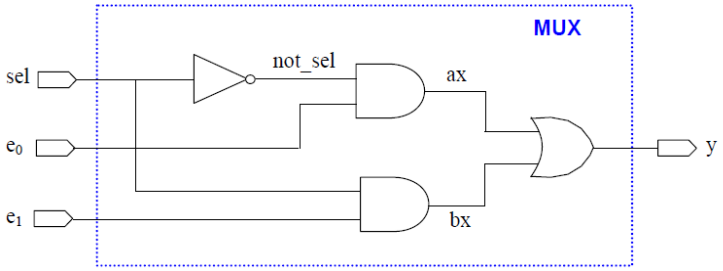
COMPORTAMIENTO

sel	y
0	e_0
1	e_1

FLUJO DE DATOS

Y = e_0 sel´ + e1 sel
not_sel = sel´
a_x = e_0 sel'
b_x = e_1 sel

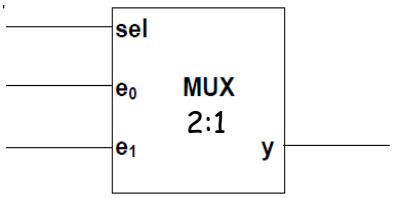
ESTRUCTURAL



4. UNIDADES DE DISEÑO (XXIV)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (III)

Con independencia del estilo de descripción empleado, se deben definir las entradas y salidas del sistema declaración de ⟶ Entidad




```
-- Los comentarios comienzan con dos guiones
-- Si se utiliza solamente datos tipo bit, se puede usar la librería "por defecto"

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux IS
        PORT (e0: IN BIT; e1: IN BIT; sel: IN BIT; y: OUT BIT
END mux;
```


4. UNIDADES DE DISEÑO (XXV)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (IV)

DESCRIPCIÓN DE COMPORTAMIENTO O ALGORÍTMICA: utiliza fundamentalmente la sentencia PROCESS, que es concurrente, sin embargo todas las sentencias contenidas en su parte descriptiva son de tipo secuencial.

- Las sentencias secuenciales se ejecutan en el mismo paso o ciclo de simulación

- Especifican los algoritmos paso a paso. El orden de escritura determina el momento de la ejecución

- Si en el dominio de una arquitectura hay más de una sentencia PROCESS, estas se ejecutan concurrentemente, sin un orden prefijado. Existe un mecanismo para controlar el orden de ejecución (no lo veremos)

4. UNIDADES DE DISEÑO (XXVI)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (V)

DESCRIPCIÓN DE COMPORTAMIENTO Sintaxis de PROCESS

[Etiqueta]: PROCESS (lista de sensibilidad) [IS]

Tipos y subtipos de datos
constantes y variables, clausula USE ficheros…..
No pueden declararse SEÑALES,
solamente elementos secuenciales

Parte declarativa

BEGIN

Sentencias secuenciales
WAIT
Asignación de señal

Parte descriptiva

END PROCESS [Etiqueta];

4. UNIDADES DE DISEÑO (XXVII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (VI)

DESCRIPCIÓN DE COMPORTAMIENTO O ALGORÍTMICA: Una sentencia PROCESS, siempre tiene que tener en su parte declarativa una sentencia WAIT, la cual causa la suspensión de la sentencia PROCESS

- En caso contario su ejecución se interpreta como un bucle infinito del que no se sale.

WAIT ON [lista de señales]

WAIT UNTIL [condición]

WAIT FOR [tiempo]

4. UNIDADES DE DISEÑO (XXVIII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (VII)

WAIT ON [lista_señales];

- Un proceso se activa cuando se produce un evento en alguna de las señales de la lista.
- Es equivalente a la sentencia WAIT, la utilización de una lista de sensibilidad entre paréntesis, junto con la palabra PROCESS. En el caso de utilizar esta estructura no puede haber otro WAIT explícito en su parte descriptiva:

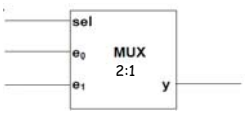
PROCESS (e_1, e_0) BEGIN sentencias secuenciales END PROCESS;	PROCESS BEGIN sentencias secuenciales WAIT ON e_1, e_0; END PROCESS;

4. UNIDADES DE DISEÑO (XXIX)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (VIII)

DESCRIPCIÓN DE COMPORTAMIENTO O ALGORÍTMICA 1: utiliza principalmente la sentencia PROCESS. Todas las sentencias están encapsuladas dentro de un PROCESS y son secuenciales, se ejecutan en el orden en que están escritas

```
ARCHITECTURE comportamiento OF mux IS
BEGIN
   PROCESS (e0,e1,sel)
   BEGIN
          IF (sel='0') THEN y<= e0;
          ELSE y<= e1;
          END IF;
   END PROCESS;
END comportamiento;
```



sel	y
0	e_0
1	e_1

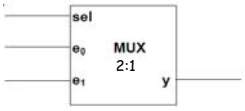
4. UNIDADES DE DISEÑO (XXX)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (IX)

DESCRIPCIÓN DE COMPORTAMIENTO O ALGORÍTMICA 2:

Dos soluciones diferentes con estilo de flujo de datos.

```
ARCHITECTURE comportamiento OF mux IS
BEGIN
        PROCESS
        BEGIN
                IF (sel='0') THEN y<= e0;
                ELSE y<= e1;
                END IF;
                WAIT ON e0,e1,sel
        END PROCESS;
END comportamiento;
```



sel	y
0	e_0
1	e_1

4. UNIDADES DE DISEÑO (XXX1)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (X)

DESCRIPCIÓN DE COMPORTAMIENTO O ALGORÍTMICA 3:
Sentencia CASE

```
[etiqueta] CASE expresión IS

        WHEN elección1 => secuencia de sentencias 1;
        WHEN elección2=> secuencia de sentencias 2;
        WHEN OTHERS => --Resto de caso
                secuencia de sentencias N;

END CASE [etiqueta]
```

4. UNIDADES DE DISEÑO (XXXII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XI)

DESCRIPCIÓN en FLUJO DE DATOS o RTL (1):

Determina el Flujo de datos entre los módulos que implementan las funciones lógicas.
La sentencia principal utilizada es **ASIGNACIÓN CONCURRENTE DE SEÑAL** y también las sentencias **BLOCK**, llamada a procedimientos….
En la **ASIGNACIÓN CONCURRENTE DE SEÑAL**, las señales no toman el valor que se les asigna de forma inmediata, sino que se actualizan sus drivers cuando se hayan ejecutado todas las asignaciones de señales consecutivas que haya en el código

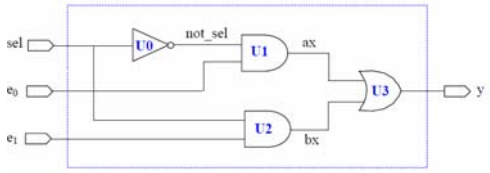
```
ARCHITECTURE flujo1 OF mux IS
BEGIN
        y<= (e0 AND (NOT sel)) OR (e1 AND sel);
END flujo1;
```


4. UNIDADES DE DISEÑO (XXXIII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XII)

DESCRIPCIÓN en FLUJO DE DATOS o RTL (2):

La sentencia principal utilizada es **ASIGNACIÓN CONCURRENTE DE SEÑAL** y también las sentencias **BLOCK**, llamada a procedimientos….
Determina el Flujo de datos entre los módulos que implementan las funciones lógicas.




```
ARCHITECTURE flujo1 OF mux IS
  BEGIN
      y<= (e0 AND (NOT sel)) OR (e1 AND sel);
END flujo1;
```

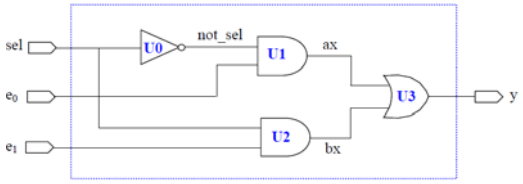
4. UNIDADES DE DISEÑO (XXXIV)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XIII)

DESCRIPCIÓN en FLUJO DE DATOS o RTL (2):

Dos soluciones diferentes con estilo de flujo de datos.

```
ARCHITECTURE flujo2 OF mux IS
-- Declaración de señales intermedias
        SIGNAL not_sel, ax, bx: BIT;
BEGIN
   not_sel<= NOT sel;
   ax<= e0 AND not_sel;
   bx<= e1 AND sel;
   y<= ax OR bx;
END flujo2
```



Operaciones intermedias

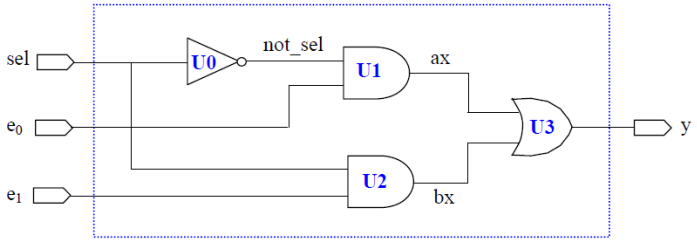
not_sel= sel'
$a_x = e_0$ sel'
$b_x = e_1$ sel

4. UNIDADES DE DISEÑO (XXXV)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XIV)

DESCRIPCIÓN ESTRUCTURAL: Se utiliza instanciación de componentes, PORT MAP y GENERATE

Es un conjunto de componentes conectados por señales (NESTLIST)
Los componentes que se utilicen han de estar previamente compilados y disponibles en la biblioteca WORK



4. UNIDADES DE DISEÑO (XXXVI)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XV)

DESCRIPCIÓN ESTRUCTURAL: DECLARACIÓN DE COMPONENTES

```
--Declaración de componentes.
--Los nombres de los componentes tienen que ser
--los mismos que los de la ENTIDAD

        COMPONENT nombre_componente_1
                PORT (locales)
        END COMPONENT nombre_componente_1;
                ……..
--Declaración de señales intermedias (internas)
--SIGNAL nombre:tipo de datos

        SIGNAL …… TIPO datos;
```

Zona declarativa: Componentes, señales

```
EJEMPLO: COMPONENT biestableD
                GENERIC(retardo: TIME);
                PORT(d, clk, clear: IN BIT; q: OUT BIT);
```


4. UNIDADES DE DISEÑO (XXXVII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XVI)

DESCRIPCIÓN ESTRUCTURAL: ESPECIFICACIÓN DE LA CONFIGURACIÓN

Permite asociar entidades y arquitecturas específicas a cada componente

- Pueden aparecer de forma implícita dentro de la propia arquitectura*

- En algunos casos no se especifica la configuración dentro de la arquitectura. En este caso se especifica la configuración:

```
CONFIGURATION nombreConfiguración OF nombreEntidad IS
```

4. UNIDADES DE DISEÑO (XXXVIII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XVII)

DESCRIPCIÓN ESTRUCTURAL: ESPECIFICACIÓN DE LA CONFIGURACIÓN

```
-- Especificación de la Configuración de componentes
-- FOR USE ENTITY
-- WORK.nombre_entidad (nombre_arquitecura)

CONFIGURATION nombre de la configuración OF nombre de la entidad IS
        USE WORK. all
        FOR nombre de la qrquitectura
                FOR nombre_componente USE ENTITY WORK (nombre de la
                     entidad (nombre dela arquitectura); END FOR;
        END FOR;
END nombre de la configuración
------------------------------
FOR ALL: componente USE ENTITY nombre_entidad (nombre_arquitectura)
```

Configuración externa

Configuración interna

EJEMPLO:

```
FOR U0: reloj1 USE ENTITY WORK reloj(reloj arq);
```


4. UNIDADES DE DISEÑO (XXXIX)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XVIII)

DESCRIPCIÓN ESTRUCTURAL: INSTANCIACIÓN DE COMPONENTES (cuerpo de la arquitectura)

```
-- Configuración de componentes
-- FOR etiqueta:nombre_componente USE ENTITY nomb_entidad (nomb_arquitect)

-- Cuerpo de la arquitectura
--Instanciación de componentes y conexionado mediante señales
   BEGIN
        U0: componente 1 PORT MAP(lista de conexiones 1);
        U1: componente 2 PORT MAP(lista de conexiones 2);
        U2: componente 3 PORT MAP(lista de conexiones 3);
   END nombre_arquitectura

        ……..

-- Con GENERIC
        U3: componente 3 GENERIC MAP (valores) PORT MAP (conexiones)
EJEMPLO: U3: inversor GENERIC MAP (retardo=>1 ns) PORT MAP (s<= e1)
```

Instanciación Conexionado

Cuerpo de la arquitectura

4. UNIDADES DE DISEÑO (XXXX)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XIX)

FICHERO PARA TEST PARA LA SIMULACIÓN

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;

-- entidad y arquitectura de test
ENTITY nombre_entidad_test IS
END nombre_entidad_test;

ARCHITECTURE nombre_arqu_test OF nombre_entidad_test IS
-- Declaración componentes
COMPONENT
Nombre_componente PORT (puertos entrada salida);
END COMPONENT;
        FOR I: nombre_componente USE ENTITY WORK nombre_entidad_componente
                (nombre_arquitec_componente);
-- Declaración de señales
        SIGNAL nombres_señales; STD_LOGIC;
        BEGIN
         I: nombre_componente POR MAP (nombres_señales);
--Drivers de las señales
        Señal_1<= ´a` AFTER x ns, ´b` AFTER y ns…….;
        Señal_2<= ´c` AFTER x ns , ´d` AFTER y ns…….;
END nombre_arqu_test;
```

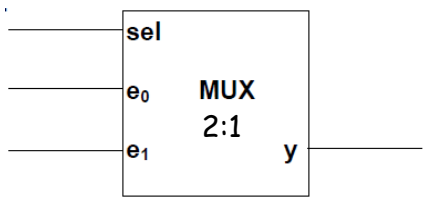
No tiene puertos

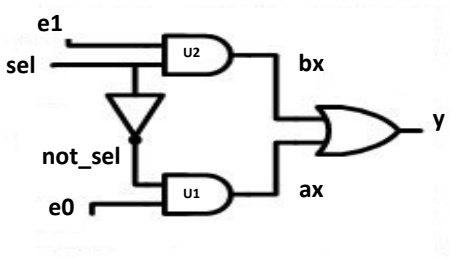
4. UNIDADES DE DISEÑO (XXXXI)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XX)

DESCRIPCIÓN ESTRUCTURAL:

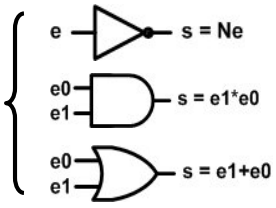
EJEMPLO: descripción en estilo estructural del multiplexor de la figura, construido con puertas





Necesitamos disponer en la librería WORK de:

- una puerta AND
- una puerta OR
- un inversor



4. UNIDADES DE DISEÑO (XXXXII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXI)

Arquitectura- *Estructural*
Modelos de componentes – INV

```
----------------------------------------
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
-- entidad y arquitectura de una puerta INV con retardo genérico
ENTITY inv IS
        GENERIC (retardo:TIME := 0 NS);
        PORT (e: IN STD_LOGIC ; -- entradas
        s : OUT STD_LOGIC -- salida);
END inv;
ARCHITECTURE flujo OF inv IS
        BEGIN
                s<= NOT e AFTER retardo;
END flujo;
```


4. UNIDADES DE DISEÑO (XXXXIII)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXII)

Arquitectura- *Estructural*
Modelos de componentes – AND

```
----------------------------
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
-- entidad y arquitectura de una puerta AND con retardo genérico
ENTITY and2 IS
        GENERIC (retardo:TIME := 0 NS);
        PORT (e1,e0 : IN STD_LOGIC ; -- entradas
        s : OUT STD_LOGIC -- salida);
END and2;
ARCHITECTURE flujo OF and2 IS
        BEGIN
                s<= e1 AND e0 AFTER retardo;
END flujo;
```

4. UNIDADES DE DISEÑO (XXXXIV)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXIII)

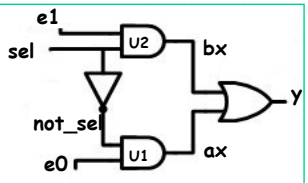
Arquitectura- *Estructural*
Modelos de componentes – OR

```
---------------------------
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
-- entidad y arquitectura de una puerta OR con retardo genérico
ENTITY or2 IS
        GENERIC (retardo:TIME := 0 NS);
        PORT (e1,e0 : IN STD_LOGIC ; -- entradas
        s : OUT STD_LOGIC -- salida);
END or2;
ARCHITECTURE flujo OF or2 IS
        BEGIN
                s<= e1 OR e0 AFTER retardo;
END flujo;
```


4. UNIDADES DE DISEÑO (XXXXV)

EJEMPLOS DE ESTILO ESTRUCTURAL (XXIV)

COMPONENT: tipos de puertas




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

--Definición de la entidad del circuito
ENTITY mux2 IS
        PORT (e0: IN BIT; e1: IN BIT; sel: IN BIT; y: OUT BIT
END mux2;

-- Definición de la arquitectura del conjunto o circuito
-- (netlist)
ARCHITECTURE estructural OF mux2 IS
   COMPONENT inv  IS GENERIC (retardo:TIME:= 0 ns);
        PORT(e:IN STD_LOGIC; s:OUT STD_LOGIC);END COMPONENT;
   COMPONENT and2 IS GENERIC (retardo:TIME:= 0 ns);
        PORT(e1,e0:IN STD_LOGIC; s:OUT STD_LOGIC);END COMPONENT;
   COMPONENT or2  IS GENERIC (retardo:TIME:= 0 ns);
        PORT(e1,e0:IN STD_LOGIC; s:OUT STD_LOGIC);END COMPONENT;
 -- SIGNAL: unen los componentes
        SIGNAL not_sel,ax,bx: STD_LOGIC;
```

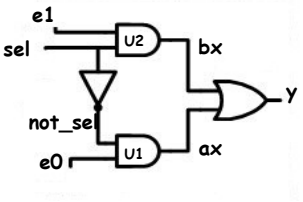
El fichero continua →

4. UNIDADES DE DISEÑO (XXXXVI)

EJEMPLOS DE ESTILO ESTRUCTURAL (XXV)

GENERIC MAP: Parámetros de los componentes (retardo para la construcción física) y PORT MAP paso de puertos

FOR y USE: enlace entre componente y entidad




```
BEGIN
        u0: inv  GENERIC MAP (1 ns) PORT MAP (e=>sel,s=>not_sel);
        u1: and2 GENERIC MAP (2 ns) PORT MAP (e1=>not_sel,e0=>e0,s=>ax);
        u2: and2 GENERIC MAP (3 ns) PORT MAP (e1=>sel,e0=>e1,s=>bx);
        u3: or2  GENERIC MAP (4 ns) PORT MAP (e1=>ax,e0=>bx,y=>sal);
END estructural;

CONFIGURATION estruc OF mux2 IS
        USE WORK.all;
        FOR estructural
                FOR ALL: inv  USE ENTITY work.inv(flujo); END FOR;
                FOR ALL: and2 USE ENTITY work.and2(flujo); END FOR;
                FOR ALL: or2  USE ENTITY work.or2(flujo); END FOR;
        END FOR;
END estruc;
```


4. UNIDADES DE DISEÑO (XXXXVII)

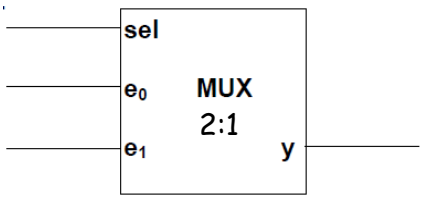
EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXVI)

```
--Declaración de componentes
--Los nombres de los componentes tienen que ser los mismos
--que los de la ENTIDAD

ARCHITECTURE estructural OF mux2 IS

        COMPONENT inv
                PORT (e: IN BIT; y: OUT BIT);
        END COMPONENT;
        COMPONENT and2
                PORT (e1,e2: IN BIT; y: OUT BIT);
        END COMPONENT;
        COMPONENT or2
                PORT (e1,e2:IN BIT ; y: OUT BIT);
        END COMPONENT;
--Declaración de señales intermedias
--SIGNAL nombre:tipo de datos

        SIGNAL ax,bx,not_sel: BIT;
```

Declaración de componentes sin retardo

4. UNIDADES DE DISEÑO (XXXXVIII)

EJEMPLOS DE ESTILO ESTRUCTURAL (XXVII)



FICHERO PARA TEST DE LA ARQUITECTURA

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;

-- entidad y arquitectura de test
ENTITY mux2_test IS
END mux2_test;

ARCHITECTURE test OF mux2_test IS
-- Declaración componentes
COMPONENT
mux2 PORT (e0, e1, sel: IN STD_LOGIC);
END COMPONENT;
        FOR I: mux2 USE ENTITY WORK mux2 (estructural);
-- Declaración de señales
        SIGNAL e0, e1, sel, y; STD_LOGIC;
        BEGIN
         I: mux2 POR MAP (e0, e1, sel, y);
--Drivers de las señales
        e0<= ´0`, ´1` AFTER 5 ns, ´0` AFTER 20 ns, ´1`AFTER 35 ns;
        e1<= ´0`, ´1` AFTER 17 ns , ´0` AFTER 38 ns;
        sel<= ´0`, ´1` AFTER 9 ns, ´0` AFTER 25 ns, ´1`AFTER 45 ns;
END test;
```

No tiene puertos

4. UNIDADES DE DISEÑO (XXXXIX)

EJEMPLOS DE ESTILO ESTRUCTURAL (XXVIII)

FICHERO DE TEST PARA LA SIMULACIÓN

mux2_tb.vhd

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;

ENTITY mux2_test IS
END mux2_test;

ARCHITECTURE test OF mux2_test IS

COMPONENT
mux2 PORT(e0,e1,c:IN STD_LOGIC;
                  sal:OUT STD_LOGIC);
END COMPONENT;

FOR I: mux2 USE ENTITY WORK.mux2(estructural);

SIGNAL e0,e1,c,sal: STD_LOGIC;

BEGIN

I:mux2 PORT MAP (e0,e1,c,sal);

e0<='0','1' AFTER 5 ns, '0' AFTER 20 ns, '1' AFTER 35 ns;

e1<='0','1' AFTER 17 ns, '0' AFTER 38 ns;

c<= '0', '1' AFTER 9 ns, '0' AFTER 25 ns, '1' AFTER 45 ns;

END test;
```

ENTITY

ARCHITECTURE

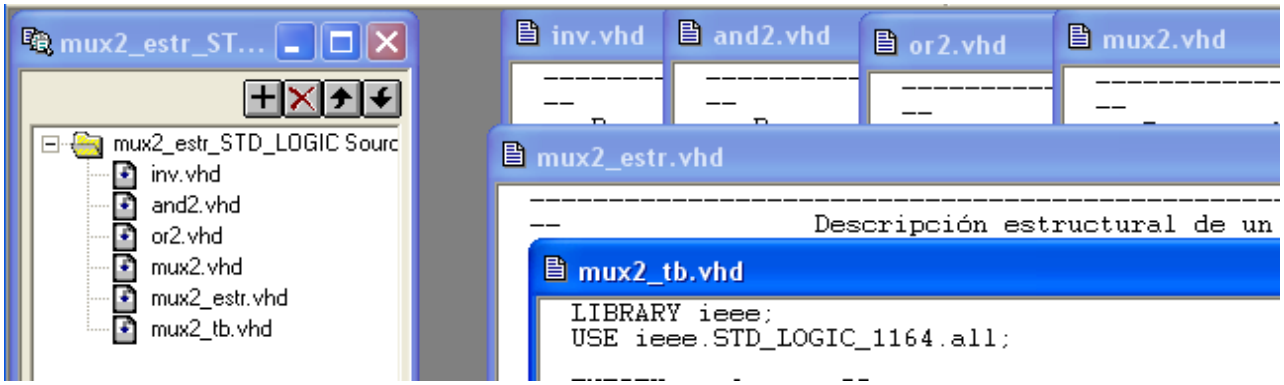
SIGNAL: entradas y salidas

DRIVERS: valores de las entradas

4. UNIDADES DE DISEÑO (L)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXIX)

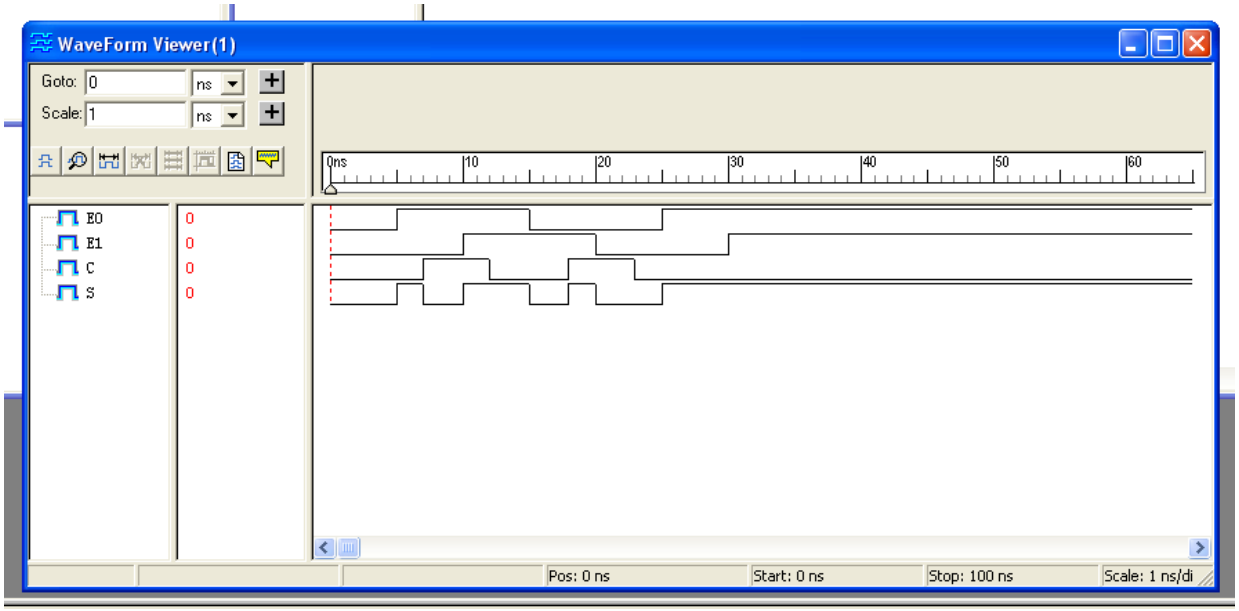
ESTRUCTURAL: Previamente se pueden definir todos los tipos de componentes necesarios en otros ficheros donde conste ENTIDAD y ARQUITECTURA (inv.vhd, and2.vhd y or2.vhd) También se necesita el Fichero de test (mux2_tex) para la visualización de la simulación.



4. UNIDADES DE DISEÑO (LI)

EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXX)

Simulación del multiplexor estructural



4. UNIDADES DE DISEÑO (LII)

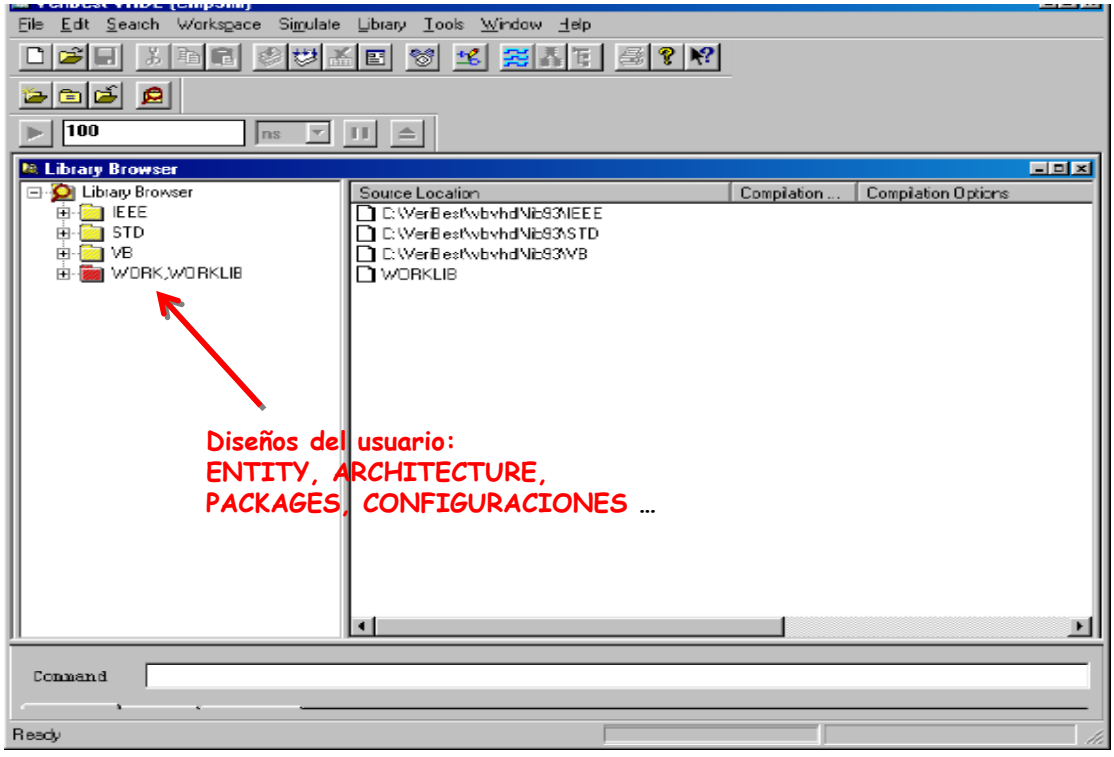
EJEMPLOS DE ESTILOS DE DESCRIPCIÓN (XXXI)

Cabecera de documentación de los ficheros

```
--De este diseño es propietario el autor y puede ser utilizado con fines de estudio

--Proyecto              Modelo de simulación en VHDL
--Diseño:               Entidad y arquitectura de "la función"
--Nombre del fichero:   Modelo_simula.vhd
--Autor:                M. Pérez Castellanos
--Fecha:                13/12/2012
--Versión:              1.0
--Resumen:              Este fichero contiene una entidad y una arquitectura
--                      en estilo flujo de datos de la función "función"-
--                      Se han utilizando tipo de datos STD_LOGIC.
--Modificaciones:
--
--Fecha          Autor           Versión         Descripción del cambio
================================================================================

================================================================================

--Definición de librerías y paquetes

LIBRARY ieee;
USE ieee STD_LOGIC_1164.all;
```

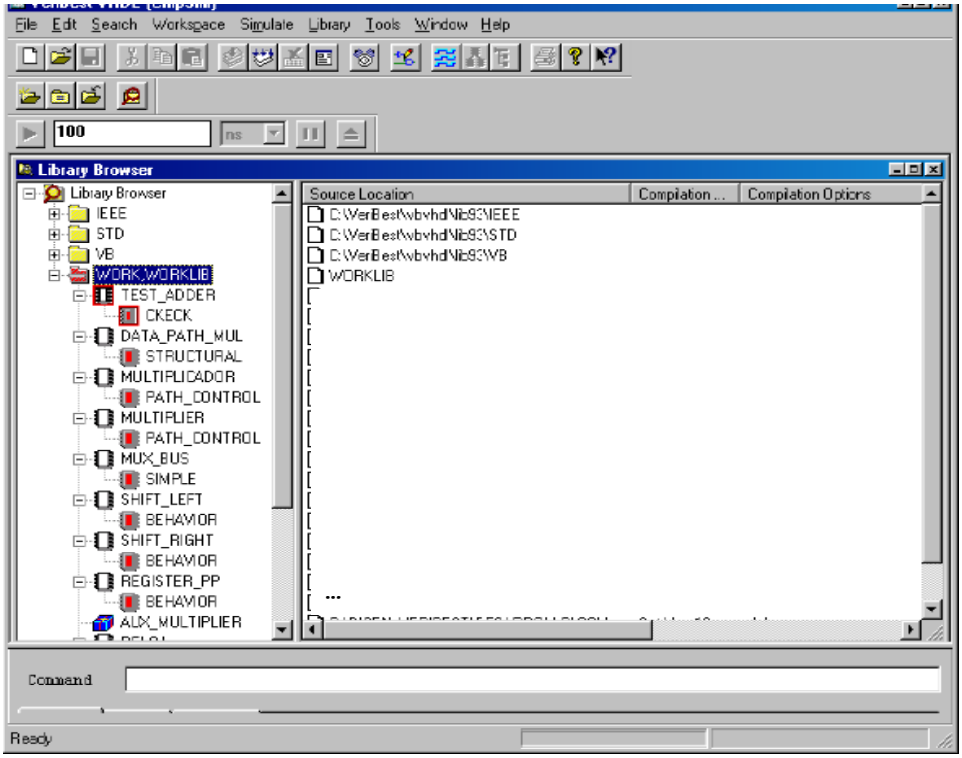

4. UNIDADES DE DISEÑO (LIII)

ESTRUCTURA DE LAS BIBLIOTECAS WORK (I)



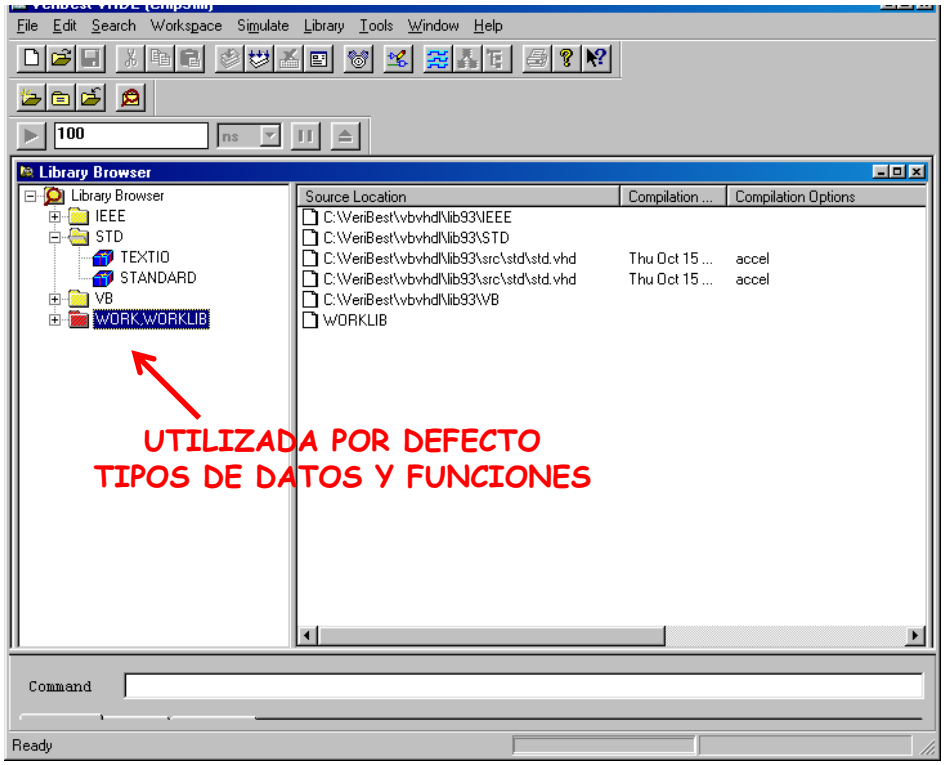
4. UNIDADES DE DISEÑO (LIV)

ESTRUCTURA DE LAS BIBLIOTECAS: WORK (II)



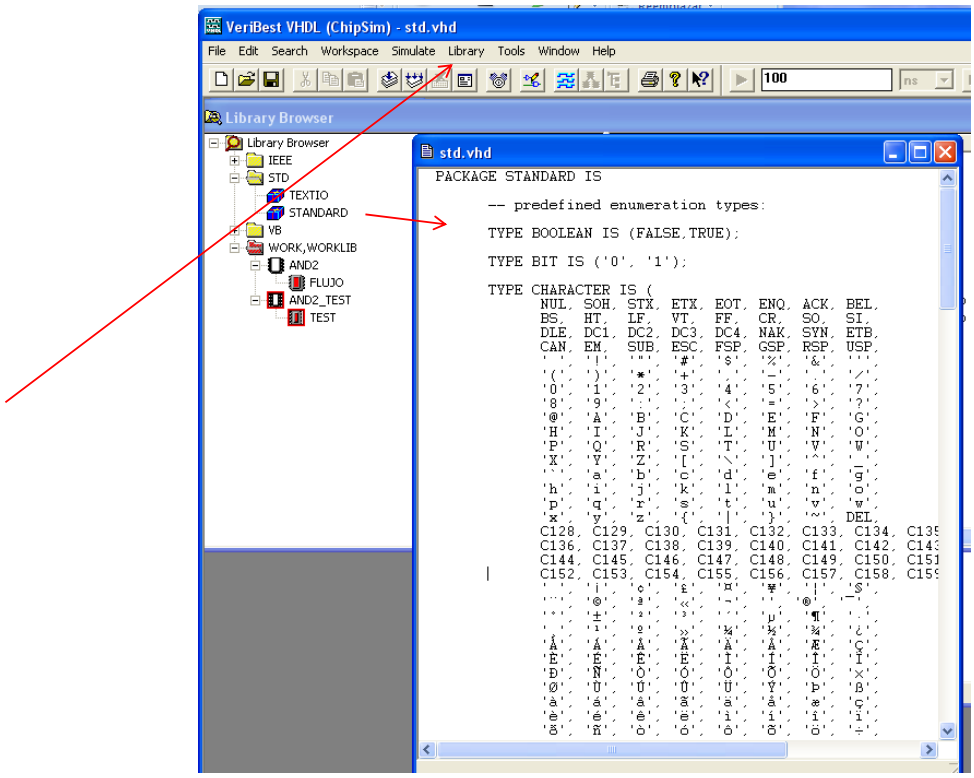
4. UNIDADES DE DISEÑO (LV)

BIBLIOTECAS (estándar) STD (I)



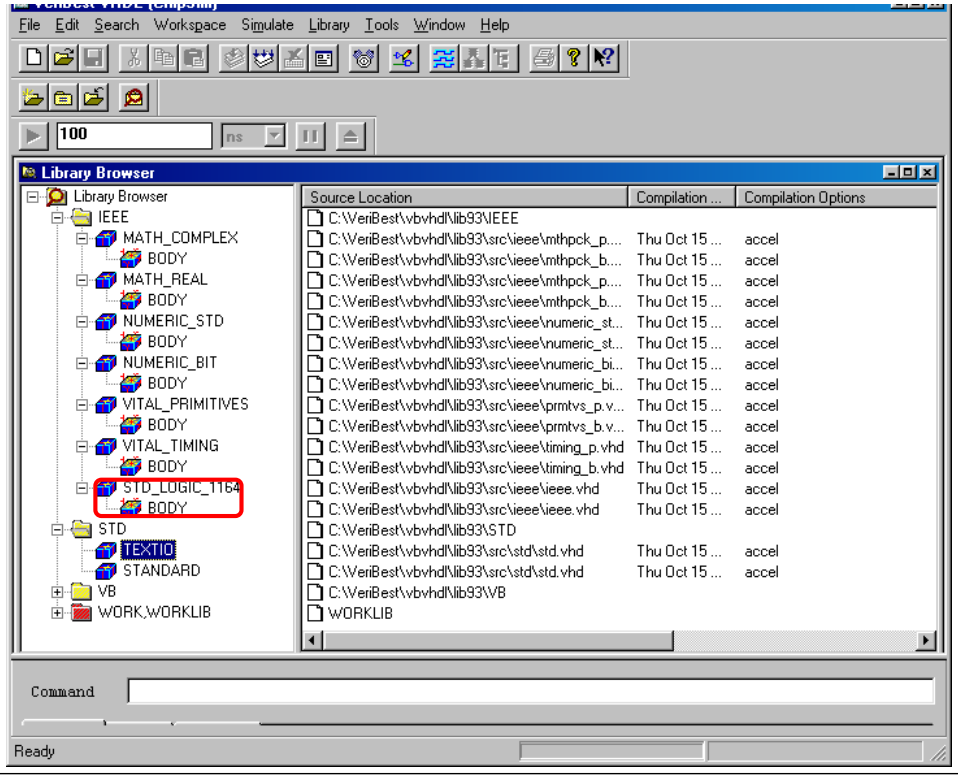
4. UNIDADES DE DISEÑO (LVI)

BIBLIOTECAS (estándar) STD (II)



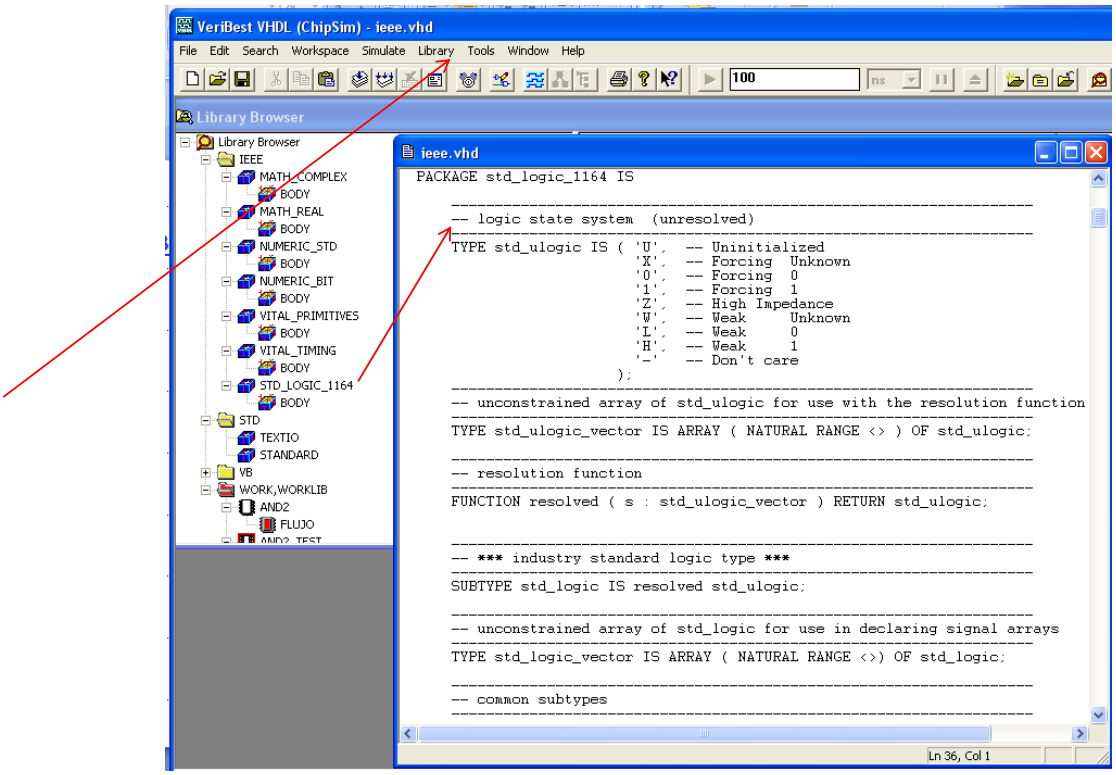
4. UNIDADES DE DISEÑO (LVII)

BIBLIOTECAS (IEEE) STD_LOGIC_1164 (I)



4. UNIDADES DE DISEÑO (LVIII)

BIBLIOTECAS (IEEE) STD_LOGIC_1164 (II)



5. TIPOS DE OBJETOS Y DATOS (I)

TIPOS DE OBJETOS	TIPOS DE DATOS
Constantes Variables Señales Ficheros	Booleano Bit Lógicos Caracter Escalares: Enumerados Enteros Físicos Coma Flotante Compuestos: Vector/matriz Registro Acceso: Punteros

5. TIPOS DE OBJETOS (I)

1. CONSTANTES

- Mantienen su valor durante toda la ejecución
- Sintaxis
  CONSTANT nombreConstante: tipoDatos[:=valorInicial];
- EJEMPLOS

  CONSTANT retardo:time:=1 ns;

  CONSTANT dirección: bit_vector:= "10001110";

  CONSTANT tabla: tipoTabla (0 TO 2):= (1,7,8);

5. TIPOS DE OBJETOS (II)

2. VARIABLES (I)

- Almacenan datos intermedios en casos donde las instrucciones se ejecutan en serie:
  - En procesos (cuando se usa PROCESS)
  - Subprogramas
- No tiene significado físico directo
  - Las asignaciones ocurren inmediatamente no tienen tiempo asociado, toman el valor que se les asigna de forma inmediata
- La sentencia de asignación de variable reemplaza el valor actual de la variable con el nuevo valor resultante de evaluar una expresión.

5. TIPOS DE OBJETOS (III)

2. VARIABLES (II)

- Sintaxis de sentencia de asignación

  VARIABLE nombreVariable: tipo[:=valorInicialOpcional];

- EJEMPLOS

  VARIABLE V1, V2:BIT:=´1´;

  Variable semaforo: tresColores:= rojo;

5. TIPOS DE OBJETOS (IV)

3. SEÑALES (I):

- Una señal mantiene una lista de valores
  - La lista incluye su valor actual y un conjunto de posible valores futuros
- Las asignaciones no son instantáneas, se actualizan en el siguiente paso de simulación (sentencias WAIT o si ha finalizado una sentencia concurrente). No actualizan sus drivers hasta que no se haya terminado de ejecutar completamente un PROCESS
- No toman el valor que se les asigna de forma inmediata
- Sirven para comunicar procesos e interconectar componentes
  - Si no se especifica un retardo (mediante sentencia AFTER), se aplica automáticamente un retardo de tipo delta.
- Deben declararse en la parte declarativa de la arquitectura y son visibles a todos los procesos y bloques de la arquitectura

5. TIPOS DE OBJETOS (V)

3. SEÑALES (II):

- Sintaxis de asignación de señal

  SIGNAL nombreSeñal: tipo[:=valorInicialOpcional];

- EJEMPLOS

  SIGNAL a: BIT:='0';

  SIGNAL busDatos: std_logic_vector (7 DOWNTO 0);

5. TIPOS DE OBJETOS (VI)

3. DIFERENCIAS ENTRE SEÑAL Y VARIABLE (I):

```
ENTITY diferenciaSenalVariable IS
        PORT(a,b:IN integer; x,y: OUT integer);
END diferenciaSenalVariable;
```

```
TEST
...
a<= 2 AFTER 1 ns;
b<= 4 AFTER 1 ns;
...
```

5. TIPOS DE OBJETOS (VII)

3. DIFERENCIAS ENTRE SEÑAL Y VARIABLE (II):

```
ARCHITECTURE conSeñal
OF diferenciaSenalVariable IS

        SIGNAL c: integer;
BEGIN
        PROCESS (a,b,c)
        BEGIN
                c<= a;
                x<= c;
                c<= b;
                y<= c;
        END PROCESS;
END conSeñal;
```

```
ARCHITECTURE conVariable
OF diferenciaSenalVariable IS

BEGIN
PROCESS (a,b)
        VARIABLE c: integer;
        BEGIN
                c:= a;
                x<= c;
                c:= b;
                y<= c;
        END PROCESS;
END conVariable;
```


5. TIPOS DE OBJETOS (VIII)

3. DIFERENCIAS ENTRE SEÑAL Y VARIABLE (III):

USO INADECUADO DE SEÑALES

```
ARCHITECTURE conSeñal OF diferenciaSenalVariable IS

        SIGNAL à,b,c,x,y: integer;
BEGIN
        PROCESS (a,b,c)
        BEGIN
                c<= a;
                x<= c+2;
                c<= b;
                y<= c+2;
        END PROCESS;
END conSeñal;
```

Valor inicial		Ejecuta	Actualiza	Ejecuta	Actualiza
a	4	4	4	4	4
b	6	6	6	6	6
c	2	4-6	6	4-6	6
x	--	2+2	4	6+2	8
y	--	2+2	4	6+2	8

Conclusión:c <= a; es superflua

Ejercicio: Este fragmento de código VHDL contiene varios errores. Indique cuatro de ellos, y explique en que consisten y como podrían subsanarse:

```
ARCHITECTURE examen OF febrero IS
  SIGNAL a, b, c: INTEGER;
  VARIABLE d: BIT;
   BEGIN
     PROCESS (a, b, d)
       BEGIN
          c <= a + d;
          b <= a +c;
  END febrero;
```

Las variables tienen que ser declaradas dentro de la parte declarativa del PROCESS

PROCESS (a, b, d)

VARIABLE d: BIT

BEGIN

PROCESS (a, b, d)

Una variable no puede formar parte de la lista de sensibilidad de un proceso, solamente pueden formar parte las señales.

PROCESS (a, b) – Comentar posibles variantes de soluciones

c <= a + d;

No es posible operar con dos tipos de datos distintos, en este caso enteros y tipo bit

Falta por cerrar el proceso

END PROCESS;

END febrero;

Al cerrar la arquitectura se ha utilizado el nombre de la entidad en lugar del nombre de la arquitectura.

END examen;

5. TIPOS DE DATOS (I)

- Cada objeto debe ser de un tipo de datos concreto
  - El tipo de datos determinará el conjunto de valores que puede tomar y las operaciones que se podrán realizar con ese objeto
- El conjunto de valores que los objetos pueden tomar no puede cambiar durante la simulación
- No hay tipos implícitos, aunque si predefinidos
  - Por ejemplo en los paquetes STD y STD_LOGIC_1164
- No hay conversiones implícitas, hay que realizarlas explícitamente

  Ejemplo erróneo: 4.0 + 3 = ¿?

5. TIPOS DE DATOS (II)

4. TIPOS DE DATOS EN BIBLIOTECAS (I):

- Predefinidos en el paquete estándar STD
  - No es necesario referenciarlos con la cláusula USE

```
-- predefined enumeration types:

      TYPE BOOLEAN IS (FALSE,TRUE);

      TYPE BIT IS ('0', '1');

      TYPE CHARACTER IS (
-- predefined numeric types:

      TYPE INTEGER IS RANGE -2147483648 TO 2147483647;

      TYPE REAL IS RANGE -1.0E38 TO 1.0E38;

   -- predefined type TIME:

      TYPE TIME IS RANGE - 2**62 -2**62 TO 2**62 - 1 + 2**62
            UNITS
                        FS;
                        PS = 1000 FS;
NS = 1000 PS;
                        US = 1000 NS;
                        MS = 1000 US;
                        SEC = 1000 MS;
                        MIN = 60 SEC;
                        HR = 60 MIN;
            END UNITS;
```


5. TIPOS DE DATOS (III)

4. TIPOS DE DATOS EN BIBLIOTECAS (II):

- Predefinidos en el paquete estándar STD
  - No es necesario referenciarlos con la cláusula USE

```
      SUBTYPE DELAY_LENGTH IS TIME RANGE 0 FS TO TIME'HIGH;

      -- function that returns the current simulation time:

      FUNCTION NOW RETURN DELAY_LENGTH;
-- predefined numeric subtypes:

      SUBTYPE NATURAL IS INTEGER RANGE 0 TO INTEGER'HIGH;

      SUBTYPE POSITIVE IS INTEGER RANGE 1 TO INTEGER'HIGH;

      -- predefined array types:

      TYPE STRING IS ARRAY (POSITIVE RANGE <>) OF CHARACTER;

      TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <>) OF BIT;

      TYPE FILE_OPEN_KIND IS (READ_MODE, WRITE_MODE, APPEND_MODE);

      TYPE FILE_OPEN_STATUS IS (OPEN_OK, STATUS_ERROR, NAME_ERROR, MODE_ERROR);

      ATTRIBUTE FOREIGN: STRING;

END STANDARD;
```

5. TIPOS DE DATOS (IV)

4. TIPOS DE DATOS EN BIBLIOTECAS (III):

- Paquete STD_LOGIC_1164
  - El paquete ha de ser llamado mediante la cláusula USE

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

- Incluye tipos de datos de tipo bit con valores extendidos ´0´,´1´, ´Z´, …..

```
std_logic

std_logic_vector (indice_1 TO/DOWNTO indice_2)
```

- Funciones de detección de flancos en señales:

```
rising_edge (identificadorSeñal)

falling_edge (identificadorSeñal)
```


6. SENTENCIAS SECUENCIALES Y CONCURRENTES (I)

SENTENCIA IF

- Selecciona la ejecución de una o ninguna de las secuencias de sentencias secuenciales dependiendo del resultado de la evaluación de una condición.
- Se recomienda para codificar prioridades

```
[etiqueta]: IF condicion THEN
              sentencias secuenciales
            (ELSIF condicion THEN
              sentencias secuenciales)
            (ELSE
              sentencias secuenciales)
            END IF [etiqueta];
```

No se recomienda el uso de IF anidados

6. SENTENCIAS SECUENCIALES Y CONCURRENTES (II)

CASE

- Selecciona una de las posibles ramas de acuerdo con el valor de una expresión.
- Todas los posibles valores de la expresión tienen que tener una opción en las ramas, puede utilizarse la palabra clave OTHERS para englobar casos iguales.
- Todas las cláusulas WHEN dentro de la sentencia CASE tienen que ser mutuamente excluyentes.
- Se prefieren a los IF
- Útiles para especificar tablas de verdad y máquinas de estados finitos FSM.

6. SENTENCIAS SECUENCIALES Y CONCURRENTES (III)

(etiqueta:) CASE expresión IS

WHEN elección = > sentencias secuenciales;

.....

WHEN elección = > sentencias secuenciales;

END CASE (etiqueta);

```
PROCESS (a,b)
  VARIABLE concatena: STD_LOGIC_VECTOR (1 DOWNTO 0);
   BEGIN
    concatena:= a & b;
    CASE concatena IS
          WHEN "00"   => s1 <= '0';
          WHEN "01"   => s1 <= '1';
          WHEN OTHERS => s1 <= '1';
    END CASE;
```

.
.
.

6. SENTENCIAS SECUENCIALES Y CONCURRENTES (VI)

Asignación de señal

Es concurrente siempre que no se incluya dentro de las sentencias de un PROCESS.

Las señales No toman el valor que se les asigna de forma inmediata, sino que no se actualizan sus drivers hasta que no se haya terminado de ejecutar completamente el PROCESO

[etiqueta:] nombre <= expresion_valor [AFTER] retardo;

Asigna valores a las señales, es equivalente a una sentencia PROCESS.

6. SENTENCIAS SECUENCIALES Y CONCURRENTES (VII)

Asignación de señal

Es concurrente siempre que no se incluya dentro de las sentencias de un PROCESS.

Asigna valores a las señales, es equivalente a una sentencia PROCESS.

```
ARCHITECTURE una OF asignación IS
        BEGIN
                A <= B+C;
END una;
```



```
ARCHITECTURE dos OF asignación IS
        BEGIN
          PROCESS (B, C)
            BEGIN
                A <= B+C;
          END PROCESS;
END dos;
```

6. SENTENCIAS SECUENCIALES Y CONCURRENTES (VIII)

Asignación condicional de señal

Equivale a un IF secuencial

SINTAXIS:

señal <= valor_expresión WHEN condicion ELSE valor_expresión;

```
ARCHITECTURE flujo_compara OF comparador IS
      BEGIN
            igual <= '1' WHEN a=b ELSE '0';
            mayor <= '1' WHEN a>b ELSE '0';
            menor <= '1' WHEN a<b ELSE '0';
END flujo_compara;
```


6. SENTENCIAS SECUENCIALES Y CONCURRENTES (IX)

Asignación de variable

- La sentencia de asignación de variable reemplaza el valor actual de la variable con el nuevo valor resultante de evaluar una expresión.

- La variable y el resultado de la expresión tienen que ser del mismo tipo.

- Las variables toman el valor que se les asigna de forma inmediata

- Las variables son similares a los objetos de otros lenguajes de programación de alto nivel.

[etiqueta:] nombre: =expresión;